Projet Jeu de la Vie

cahier des charges

I – Présentation du projet

1 - Introduction

Le produit à réaliser est un logiciel graphique simulant le *Jeu de la Vie* : l'évolution de populations de cellules dans un espace en fonction de leur environnement, selon des règles strictes.

Le logiciel permettra d'automatiser les calculs de l'évolution et de visuliser les changements dans la population en temps réel.

2 – Détails sur le Jeu de la Vie

L'espace n'a que deux dimensions et est une grille finie de cases. Les cellules sont fixes, chacune placée dans une case. Cache case peut contenir une cellule d'une population donnée ou bien être vide.

L'unité temporelle élémentaire du *Jeu de la Vie* est le *cycle*. A chaque cycle, pour chaque case de l'état actuel on détermine ce que contiendra cette case à l'état suivant selon un jeu de règles. L'évolution est donc simultanée pour toutes les cellules.

Les règles d'évolution associent une action à une condition sur l'environnement de la case traitée. Par exemple, une règle pourrait être « Si il y a plus de 3 cellules bleues dans un rayon de 1 case, une cellule rouge nait ». Les actions possibles d'une règle sont *naissance*, *vie* et *mort*. Si la condition d'une règle est vérifiée, l'action est appliquée (si applicable : on ne peut pas tuer une case vide, par exemple) et les règles suivantes sont ignorées. Le *rayon* désigne ici une *norme infinie*, plus adaptée que la norme euclidienne pour une grille.

3 – Fonctionnalités principales

L'utilisateur pourra fournir son propre jeu de règles à l'exécution du programme. Les règles devront être rédigées dans une certaine syntaxe, décrite en annexe.

Par défaut, l'état intial est aléatoire : la grille est aléatoirement remplie de cellules de diverses populations. L'utilisateur pourra fournir son propre état initial à l'exécution du programme. Le format de données associé est discuté en annexe.

4 – Fonctionnalités secondaires

Ces fonctionnalités ne sont pas nécessaires et n'apporteraient qu'un léger confort d'utilisation.

On pourra permettre à l'utilisateur de mettre le programme en pause (de mettre l'évolution en pause) ou de sauvegarder l'état actuel du système.

On pourra aussi fournir à l'utilisateur une interface graphique pour la saisie de l'état initial et/ou des règles.

Enfin, on pourra fournir un moyen à l'utilisateur de modifier dynamiquement les règles et/ou l'état

du système. Remarque : cette fonctionnalité est plus puissante que la précédente, si on permet de mettre le jeu en pause alors la précédente fonctionnalité devrait être complètement remplacée par celle ci.

II – Ressources de développement

1 – Equipe de développement

Il n'y aura qu'une seule personne dans l'équipe. Cette personne aura à la fois le rôle de chef de projet, de développeur et de testeur.

2 – Outils de développement

On développera le logiciel dans le langage Java. La version de Java visée est la 1.6.

Le développement se fera sur un système Linux équippé d'une machine virtuelle Java compatible Java 1.6.

L'environnement de programmation pourra être Eclipse ou bien un éditeur de texte et un makefile.

La documentation, quel que soit le format, devra être convertible en pdf et LaTeX.

Etant donné le nombre de développeurs, il ne sera pas nécessaire d'utiliser un système de versions concurrentes (CVS, SVN...). A la place, on pourra simplement créer une archive de chaque release.

III – Description du système

1 – Aspects du système

D'après les fonctionnalités, les principaux éléments du système sont :

- La fenêtre principale, affichant la grille
- La grille et l'état actuel
- Le jeu de règles fournies par l'utilisateur
- Un analyseur lexical et syntaxique, pour lire les règles fournies par l'utilisateur
- Un algorithme qui applique les règles
- Un procédé de dessin de la grille sur la fenêtre

2 – Conception interne

L'analyseur lexical et syntaxique ne sert qu'au tout début de l'exécution, afin d'initialiser les règles. Il n'apparait donc pas dans le schéma d'architecture (fig. 1) qui ne traite pas l'évolution temporelle du programme.

Le jeu de règle, l'état actuel et la fenêtre sont des données indépendantes manipulées par les procédés de dessin et d'évolution à chaque cycle.

En pratique, on veillera à procéder au dessin juste après l'évolution, afin de donner à l'utilisateur une image en temps réel et non en différé. Aussi, le procédé d'évolution travaillera sur des données temporaires et ne modiefira l'état actuel qu'à la fin, afin de ne pas causer d'effets de bord (rappelons que les conditions des règles dépendent d'un bloc de cases de l'état actuel).

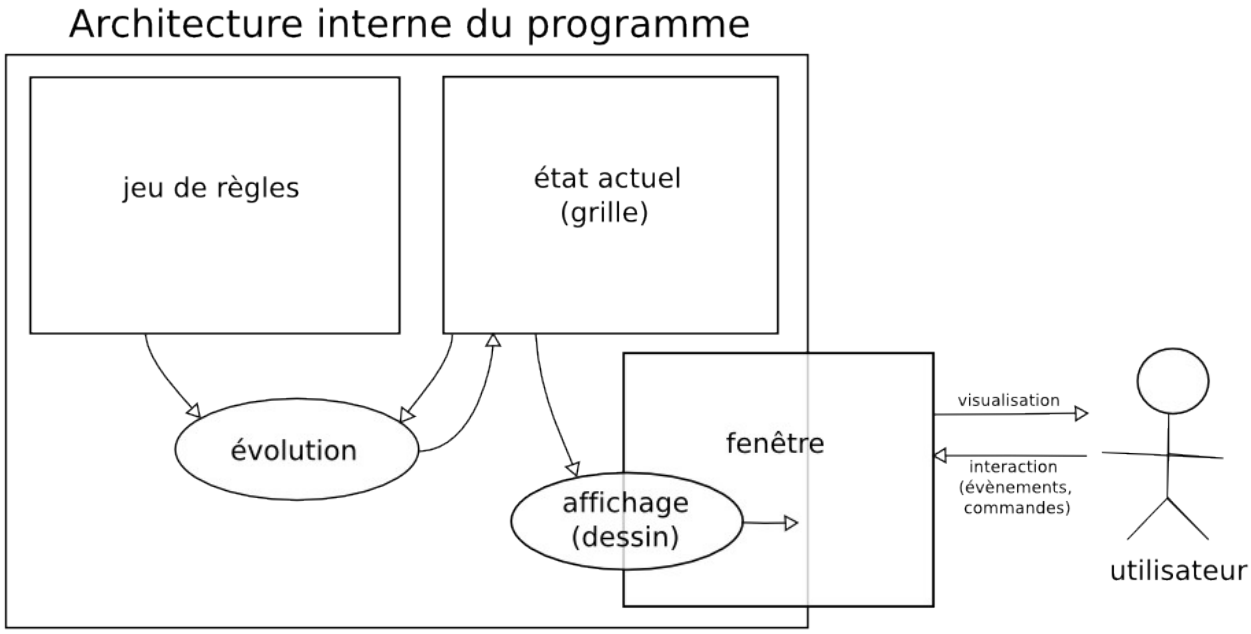

Figure 1: architecture interne du programme

IV – Documents et support

1 – Documents fournis à l'utilisateur

- Manuel d'installation (en français)
- Manuel d'utilisation (en français)

2 – Support pour l'utilisateur

Le support sera assuré par une autre équipe, certainement constituée d'exactement 0 personnes.

En ce sens, on garantit qu'aucun support ne sera fourni à l'utilisateur.

V – Contraintes d'utilisation

1 – Contraintes techniques

Le programme ne fonctionnera que sur une plateforme Java compatible Java 1.6.

L'exécution du programme suppose la présence d'un environnement graphique fenêtré.

Le manuel d'installation ainsi qu'une partie du manuel d'utilisation (traitant de l'utilisation en ligne de commande) supposeront un système Linux équipé d'une machine virtuelle.

2 – Contraintes légales (licences)

Le programme sera distribué sous Licence LGPL, à l'exception de la classe principale, exécutable, qui sera sous licence GPL.

VI – Annexes

1 – Règles d'évolution

La syntaxe des règles est assez simple. On associe une action à une condition sur l'environnement. Chaque population est nommée par une lettre de l'alphabet.

On écrira les éléments fixes de syntaxe (les constantes) en gras et les opérateurs syntaxiques (?, +,*...) en rouge. On emploiera aussi l'opérateur ...() représentant une liste d'éléments séparés par une certaine constante.

Syntaxe `rule` : `rule-condition -> action (population)`

Exemple : `3 [a = 2] -> naissance(b)`
Effectue : si dans un rayon de 3 il y a exactement 2 cellules de la population 'a' alors naissance d'une cellule de la population 'b'.

La sous-syntaxe `action` ne peut être qu'un des mots clés suivants : `naissance`, `vie` ou `mort`.

La sous-syntaxe `population` est une lettre de l'aphabet.

La sous-syntaxe `rule-condition` est plus complexe : on peut composer plusieurs rule-condition avec des opérateurs ET (&), OU (|), NON (!). Par soucis de lisibilité, on utilise une sous-syntaxe pour chaque niveau de priorité d'opérateur.

```
rule-condition : and-rule-condition...(|)

and-rule-condition : basic-rule-condition...(&)

basic-rule-condition : radius [condition]
                     | !basic-rule-condition
```

La sous-syntaxe `radius` est un nombre entier positif.

La sous-syntaxe `condition` peut elle aussi être combinée avec des opérateurs booléens mais on peut aussi parenthéser une sous-condition, ce qui rend cette sous-syntaxe récursive.

```
condition : and-condition...(|)

and-condition : basic-condition...(&)

basic-condition : comparison
                | (condition)
                | ! (condition)

comparison : population comparison-operator constant
```

La Sous-syntaxe `comparison-operator` est un opérateur de comparaison parmis (**<**, **<=**, **=**, **>**, **>=**, **!=**).

La sous-syntaxe `constant` est une constante entière positive.

Un fichier de syntaxe fourni par l'utilisateur contient la déclaration des populations suivie d'une ou plusieurs règles :

```
syntax-file : populations={population...(,)} rule+
```

2 – Fichier de grille

Un fichier de grille est un simple fichier texte contenant un état du jeu de la vie à un instant donné (la grille de cases). Une ligne du fichier correspond à une ligne de la grille du programme. Le fichier doit contenir des espaces et/ou des lettres associées aux populations déclarées dans le fichier de règles.